FALCONE81
Falcone, Joseph R.; A Vector Programming Language; ACM SIGPLAN Notices (April 1981) pp. 36-50.

A vector processor is any computer which meets the following criterion:

```
for  T(1) = time for a scalar operation
     T(n) = time for a vector operation on n operands
 and T(1) approx.= T(n) for most n of interest
```

Languages such as FORTRAN and ALGOL have been extended to encompass the special vector processor capabilities. This is a poor approach as certain of the vector processor instructions can not be utilized without compromising the statement or expression syntax of the language. A "vectorising" compiler can be used to attempt to extract parallelism from serial coding, but results are disappointing due to the limited capabilities of such a compiler. It turns out that some of the vector processor's instructions are more powerful (or high level) than the high level language's instructions. Languages have been designed specifically to deal with vector processors, but the author wanted to design a language for vector processors that had the benefits of a language of the ALGOL family. Some of the design issues were typing, abstraction, genericism, and goto-less programming. The language developed was given the name FLAN.

FLAN's type structure is similar to that of Pascal with types called "groups" being very similar to enumerated types. FLAN of course allows operations on whole arrays and has a large collection of operators to facilitate sophisticated vector computation. Array sizes must be equivalent unless it is specifically stated otherwise. Operations between an array and a scalar are legal with the scalar used as the operand in each of the vector operations. Procedures are handled in a fairly standard manner and the abstraction facility of modules is provided (inherited from Modula).

The author points out that the goto statement is not included in the language for all the standard reasons, but that there is an additional problem peculiar to vector processors. He points out that comparison operations on whole arrays lead to problems of the following type:

```
if a[1..50] > b[1..50]
    then goto here
       else goto there
```

This leads to possibly many distinct instruction streams. But his answer to this problem is a loop structure with generalised exit statements.

```
BEGIN
    . . .
    WHEN finished LEAVE area WITH cleanup;
    . . .
END area;
```

But this does not seem to solve the problem as the "finished" part could easily be a comparison statement on two arrays. Thus it seems that the problem is that of directing flow of control with comparisons on whole arrays, not the use of gotos.

FLAN has three assignment statements: direct, interchange and conditional. The interchange statement is added because many computers now have an interchange machine instruction. The conditional assignment does not really belong in this type of language, and its syntax is awkward:

```
x <-? y > z then y
             else z
```

rather than

```
if y > z then x <- y
          else x <- z
```

In the section on control structures, the author again uses arrays in the conditional part of an if statement without addressing the problem of creating two distinct instruction streams. It is not clear what amount of parallellism is allowed on a vector processor and how the language deals with this problem. The section on control structures is also a weak one as a result of the examples used. Some of the examples have mistakes in them and some are either misleading or have mistakes in them.

In the conclusion, the author states his goal as being the design of a language to be used by people rather than just by scientists. I believe that FLAN could be a reasonable realization of that aim, but much is left unclear in the paper regarding the control flow and the degree of parallellism available.